

# AN ENERGY-EFFICIENT AND MASSIVELY PARALLEL APPROACH TO VALID NUMERICS

---

John L. Gustafson, Ph.D.

*CTO, Ceranovo*

*Director, Massively Parallel Technologies, Etaphase,  
Clustered Systems*



# Big problems facing computing

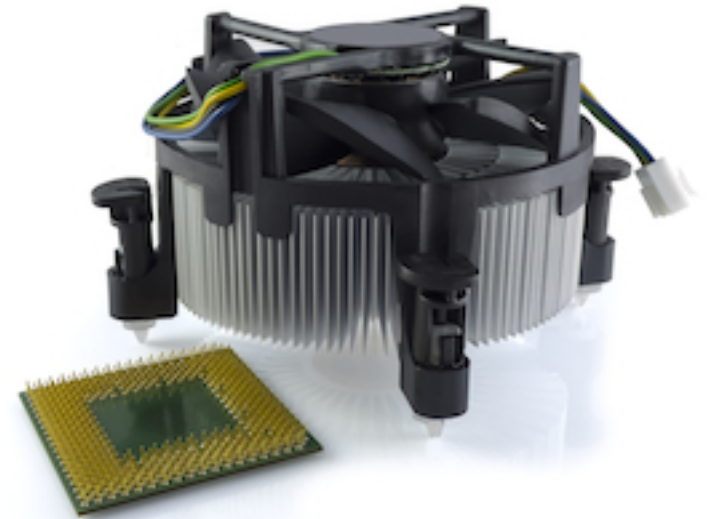
- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use
- Not enough bandwidth (the “memory wall”)
- Rounding errors more treacherous than people realize
- Rounding errors prevent use of parallel methods
- Sampling errors turn physics simulations into guesswork
- Numerical methods are hard to use, require experts
- IEEE floats give different answers on different platforms

# The ones *vendors* care most about

- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use
- Not enough bandwidth (the “memory wall”)
- Rounding errors more treacherous than people realize
- Rounding errors prevent use of parallel methods
- Sampling errors turn physics simulations into guesswork
- Numerical methods are hard to use, require experts
- IEEE floats give different answers on different platforms

# Too much power and heat needed

- Huge heat sinks
- 20 MW limit for exascale
- Data center electric bills
- Mobile device battery life
- Heat intensity means *bulk*
- *Bulk* increases *latency*
- *Latency* limits *speed*



# More parallel hardware than we can use

- Huge clusters usually partitioned into 10s, 100s of cores
- Few algorithms exploit millions of cores except LINPACK
- *Capacity* is not a substitute for *capability*!



# Not enough bandwidth (“Memory wall”)

Operation	Energy consumed	Time needed
64-bit multiply-add	200 pJ	1 nsec
Read 64 bits from cache	800 pJ	3 nsec
Move 64 bits across chip	2000 pJ	5 nsec
Execute an instruction	7500 pJ	1 nsec
<i>Read 64 bits from DRAM</i>	<b>12000 pJ</b>	<b>70 nsec</b>

Notice that 12000 pJ @ 3 GHz = 36 watts!

One-size-fits-all overkill 64-bit precision *wastes energy, storage bandwidth*

# Happy 100<sup>th</sup> Birthday, Floating Point

1914: Torres proposes automatic computing with a fraction and a scaling factor.

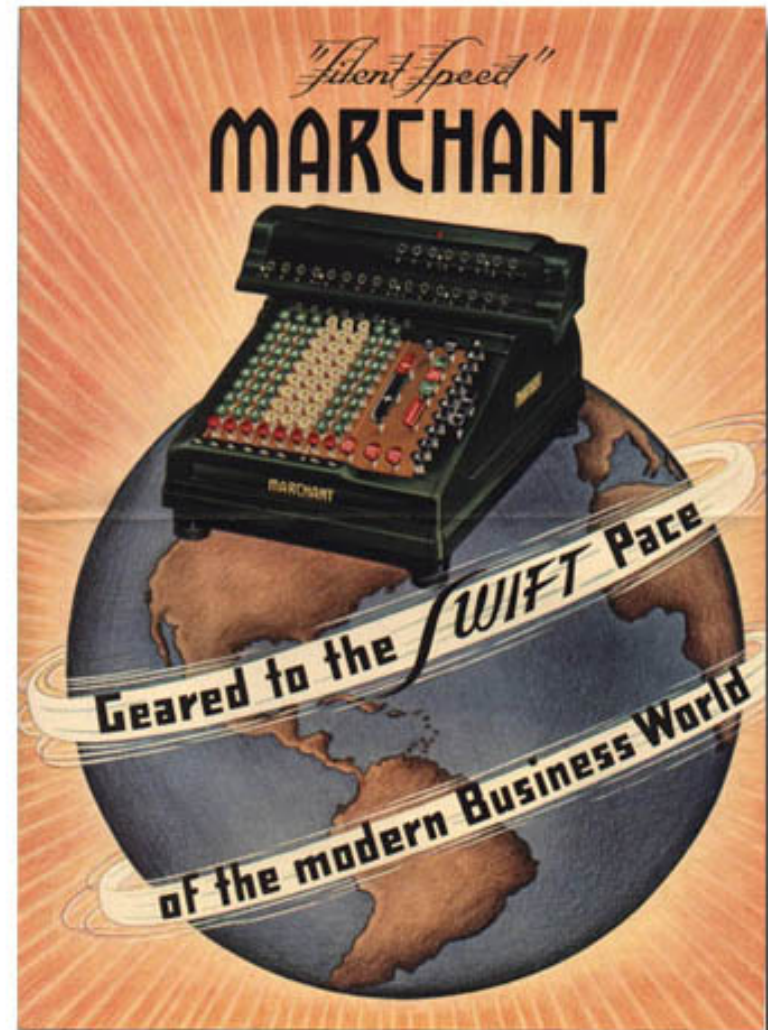
2014: We still use a format designed for World War I hardware capabilities!





# Floats designed for *visible* scratch work

- OK for *manual* calculations
  - Operator sees, remembers errors
  - Can head off overflow, underflow
- Automatic math *hides* all that
- No one sees processor “flags”
- Disobeys algebraic laws
- Wastes bit patterns as NaNs
- IEEE 754 “standard” is really the IEEE 754 *guideline*; optional rules spoil consistent results





# Analogy: Printing in 1970 vs. 2014

1970: 30 sec per page

```
DISK OPERATING SYSTEM/360 FORTRAN 360N-FD-451 CL
C ROBERT GLASER, RANDALLSTOWN SENIOR, GROUP A, P AND S
C PRIME NUMBERS
DO 100 I=1,1000
  J=2
  K=2
  2 L=J*K
  IF (L-I) 10,100,10
10 M=2+3
  IF (K-I) 20,3,3
20 K=K+1
  GO TO 2
  3 K=2
  IF (J-I) 5,4,4
  5 J=J+1
  GO TO 2
  4 WRITE (3,6) I
  6 FORMAT (I10)
100 CONTINUE
STOP
END
```

2013: 30 sec per page



Faster technology is for *better* prints,  
not thousands of low-quality prints per second.  
Why not do the same thing with computer arithmetic?

This is just... sad.

Subtotal:	\$64.99
Sales Tax:	\$4.71
<hr/>	
TOTAL	\$69.6999999999999999
Total Items Picked Up Is:1	
Customer Signature:-----	
By signing, you acknowledge you have	

# Floats *prevent use of parallelism*

- No associative property for floats
- $(a + b) + (c + d)$  (parallel)  $\neq ((a + b) + c) + d$  (serial)
- Looks like a “wrong answer”
- Programmers trust serial, reject parallel
- IEEE floats report rounding, overflow, underflow in *processor register bits that **no one ever sees**.*

# A New Number Format: The Unum

- Universal **num**bers
- **Superset** of IEEE types, both 754 and 1788
- Integers  $\rightarrow$  floats  $\rightarrow$  unums
- No rounding, no overflow to  $\infty$ , no underflow to zero
- They obey algebraic laws!
- Safe to parallelize
- *Fewer bits* than floats
- But... they're *new*
- Some people don't like *new*



“You can’t boil the ocean.”

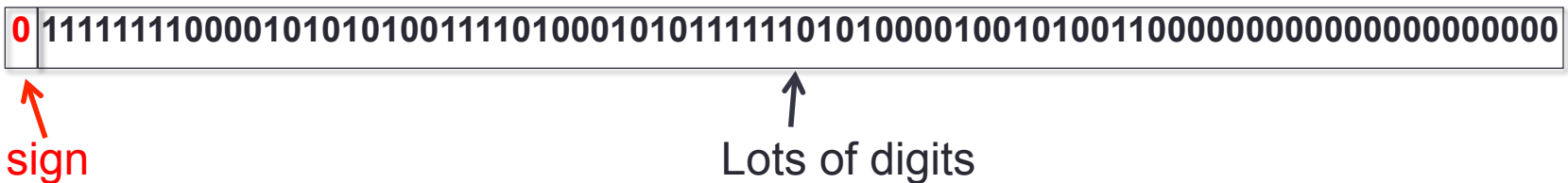
—Former Intel exec, when shown the unum idea



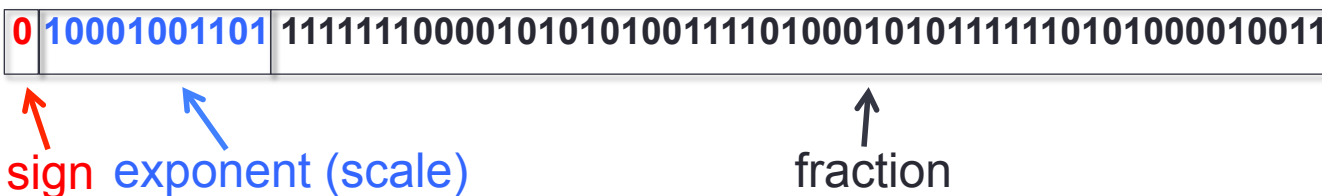
# Three ways to express a big number

Avogadro's number:  $\sim 6.022 \times 10^{23}$  atoms or molecules

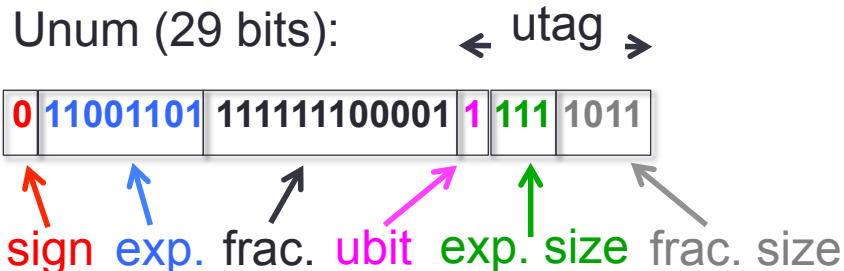
### Sign-Magnitude Integer (80 bits):



## IEEE Standard Float (64 bits):



Unum (29 bits):



*Self-descriptive “utag” bits track and manage uncertainty, exponent size, and fraction size*

# Why unums use fewer bits than floats

- Exponent smaller by about 5 – 10 bits, typically
- Trailing zeros in fraction compressed away, saves ~2 bits
- Shorter strings for more common values
- Cancellation removes bits and the need to store them

*IEEE Standard Float (64 bits):*

0	10001001101	1111111000010101010011110100010101111110101000010011
---	-------------	--



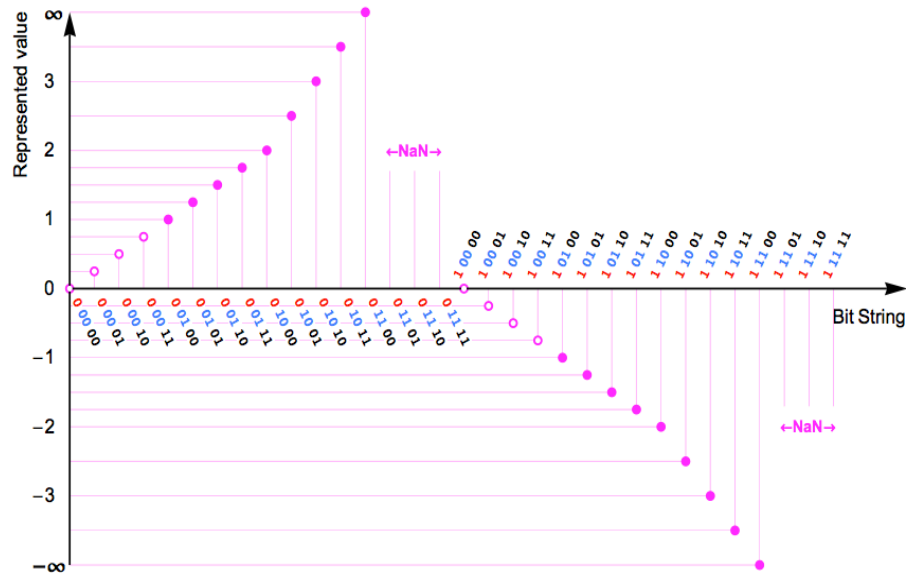
*Unum (29 bits):*

0	11001101	111111100001	1	111	1011
---	----------	--------------	---	-----	------

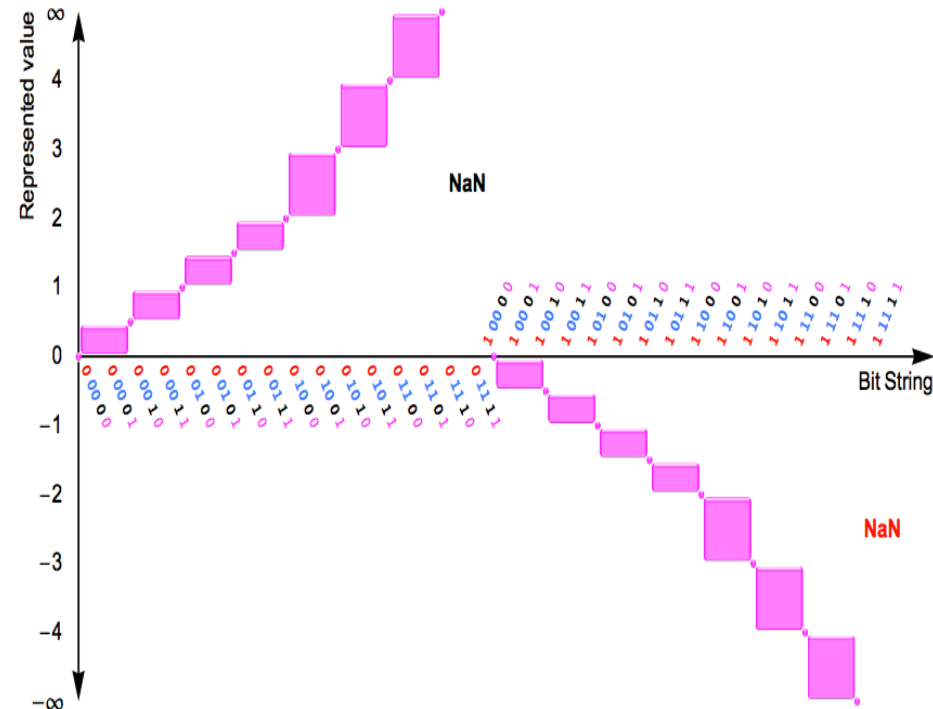


# Open *ranges*, as well as exact points

Bit string meanings  
using IEEE Float rules



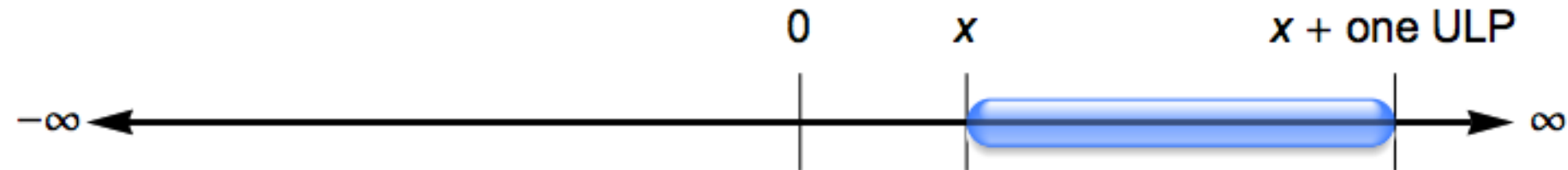
Bit string meanings  
in unum format



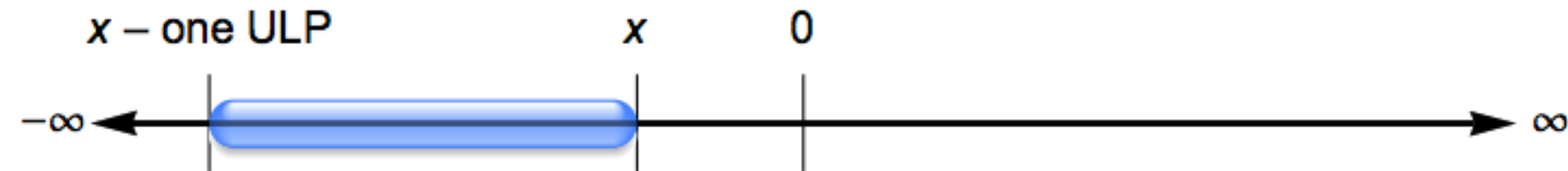
Complete representation of *all* real numbers using a finite number of bits

# Ubounds are the hull of 1 or 2 unums

Open Interval Represented by a Positive Inexact Unum

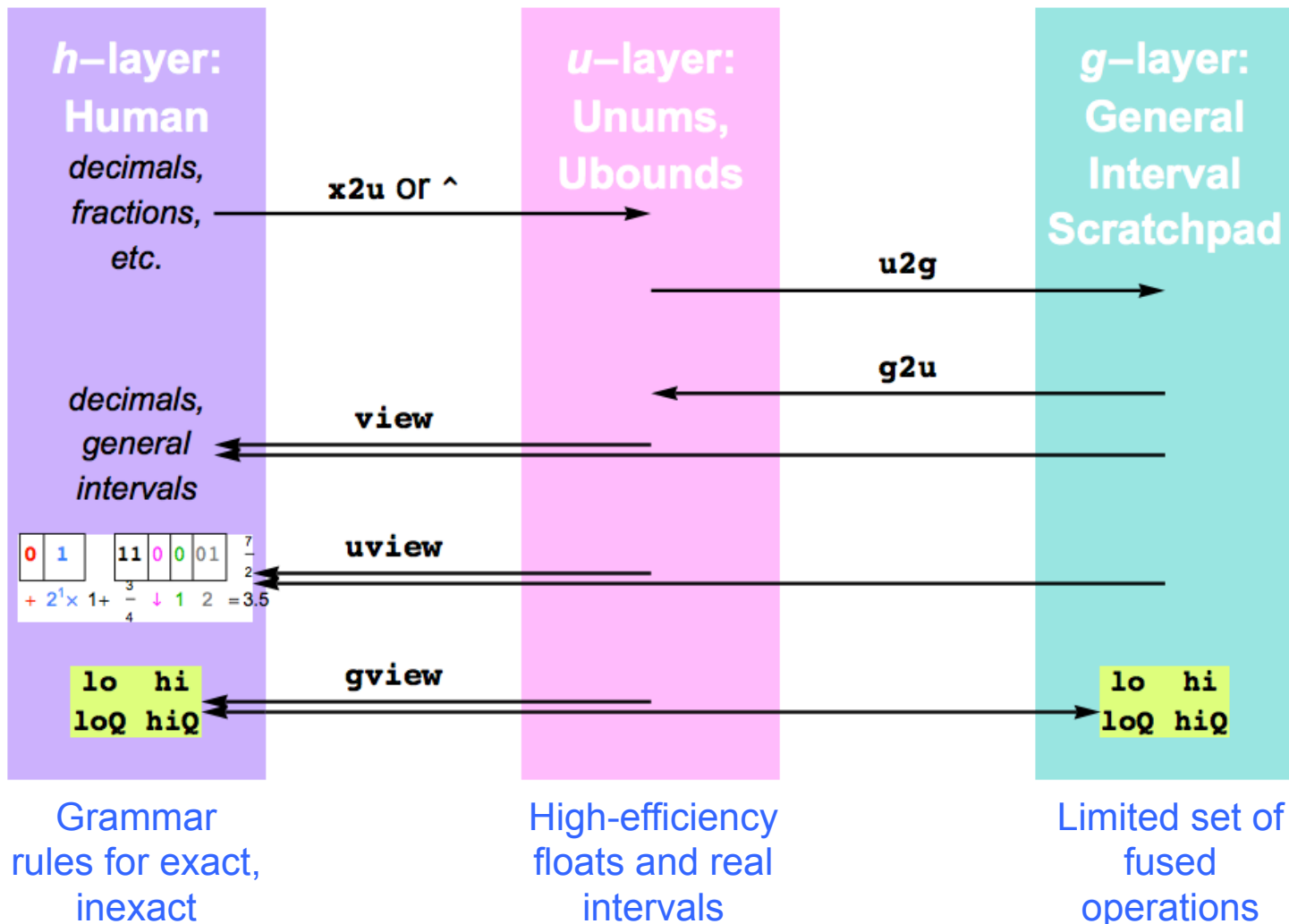


Open Interval Represented by a Negative Inexact Unum



- Includes closed, open, half-open intervals
- Includes  $\pm\infty$ , empty set, quiet and signaling NaN
- *Unlike traditional intervals, ubounds are closed and **lossless** under set operations*

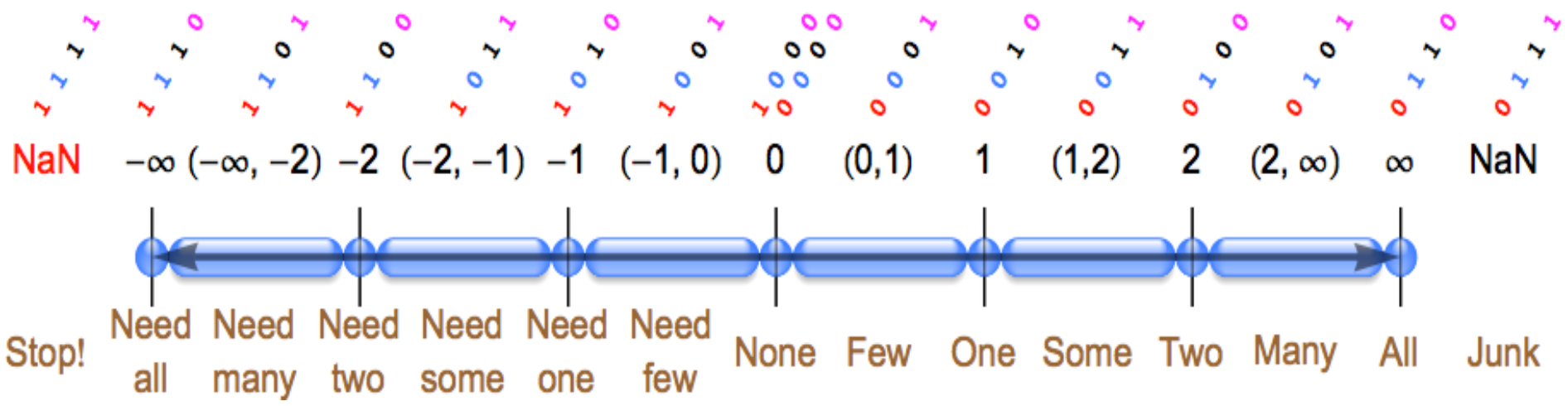
# The three layers of computing



# The Warlpiri unums

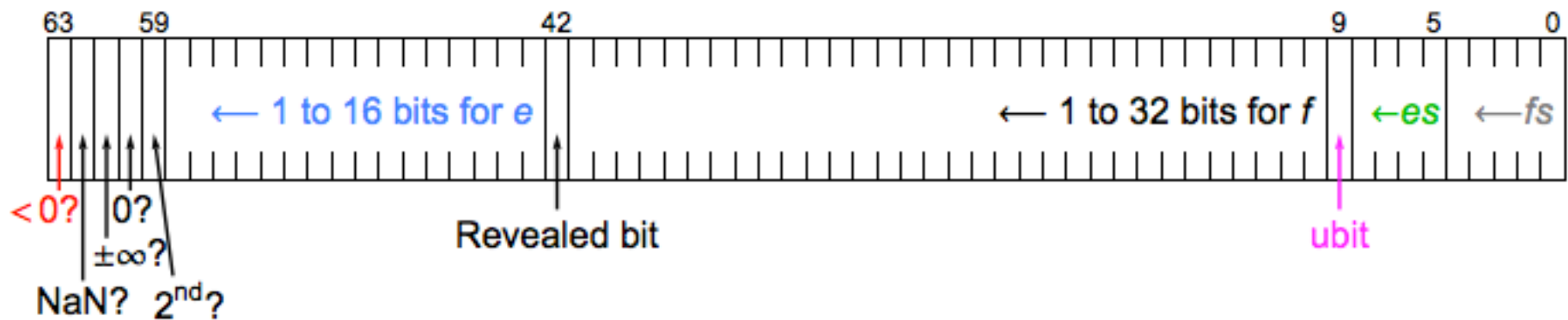
Before the aboriginal Warlpiri of Northern Australia had contact with other civilizations, their counting system was “One, two, many.”

Maybe they were onto something.

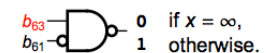
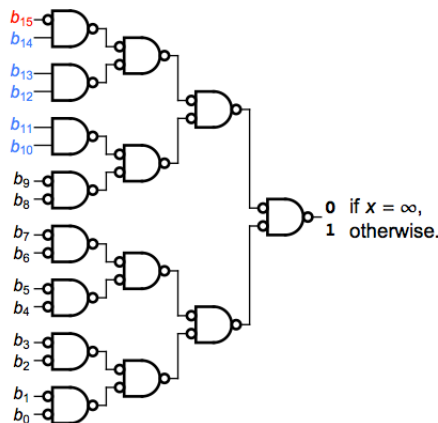


# Fixed-size unums: faster than floats

- Warlpiri ubounds are one byte, but closed system for reals
- Unpacked unums pre-decode exception bits, hidden bit



Circuit required for  
“IEEE half-precision  
float =  $\infty$ ?”



Circuit required for  
“unum =  $\infty$ ?”  
(any precision)

# Floating Point II: The Wrath of Kahan

- Berkeley professor William Kahan is the father of modern IEEE Standard floats
- Also the authority on their many dangers
- Every idea to fix floats faces his tests that expose how new idea is *even worse*



*Working unum environment  
completed August 13, 2013.*

*Can unums survive the  
wrath of Kahan?*



# A Typical Kahan Challenge

“Define functions with:  $E(0) = 1$ ,  $E(z) = \frac{e^z - 1}{z}$ .  $Q[x] = \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2 + 1}}$ .  $H(x) = E(Q(x))^2$ .

Compute  $H(x)$  for  $x = 15.0, 16.0, 17.0, 9999.0$ . Repeat with more precision, say using BigDecimal.”

- Correct answer: (1, 1, 1, 1).
- IEEE 32-bit: (0, 0, 0, 0) **FAIL**
- IEEE 64-bit: (0, 0, 0, 0) **FAIL**
- Myth: “Getting the same answer with increased precision means the answer is correct.”
- IEEE 128-bit: (0, 0, 0, 0) **FAIL**
- Extended precision math packages: (0, 0, 0, 0) **FAIL**
- Interval arithmetic: Um, somewhere between  $-\infty$  and  $\infty$ . **EPIC FAIL**
- Unums, **6-bit** average size: (1, 1, 1, 1) **CORRECT**

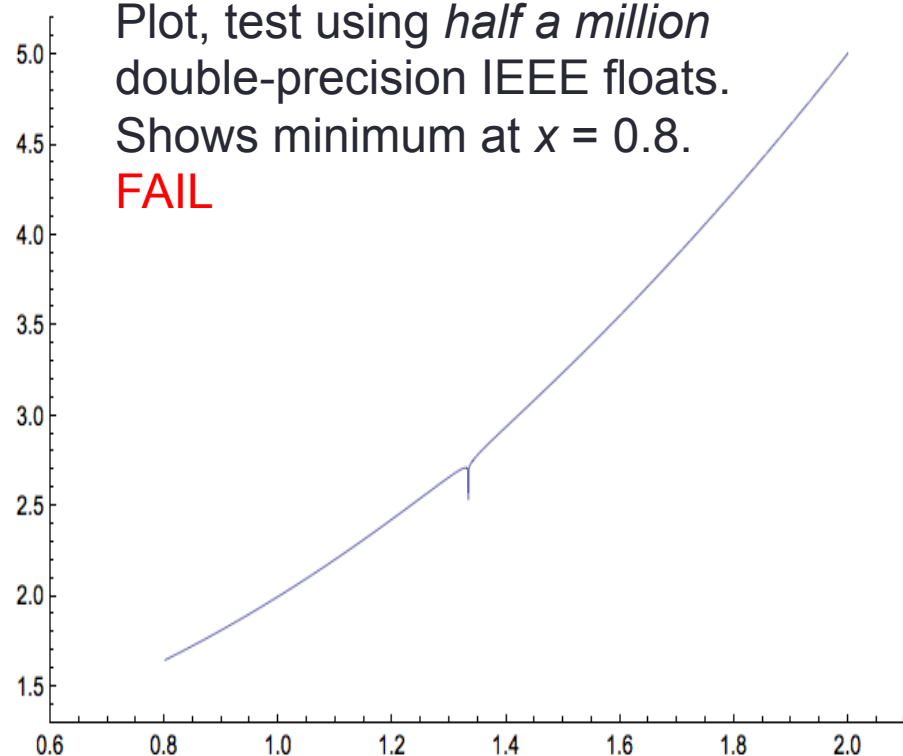
**I have been unable to find a problem that “breaks” unum math.**

# Kahan's "Smooth Surprise"

Find minimum of  $\log(|3(1-x)+1|)/80 + x^2 + 1$  in  $0.8 \leq x \leq 2.0$

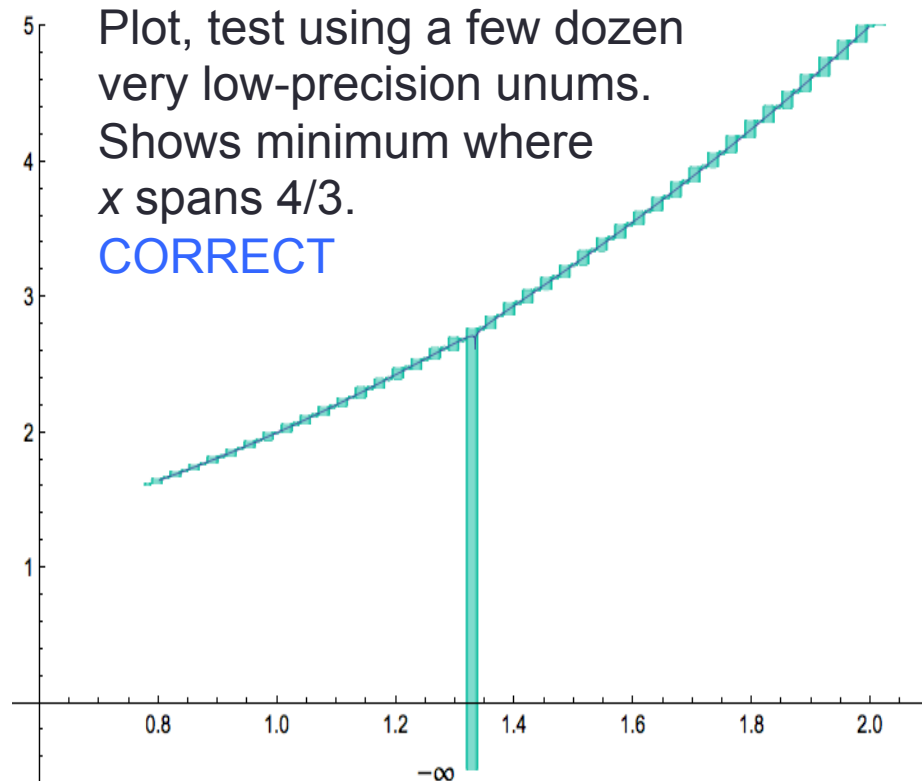
Plot, test using *half a million* double-precision IEEE floats. Shows minimum at  $x = 0.8$ .

**FAIL**



Plot, test using a few dozen very low-precision unums. Shows minimum where  $x$  spans  $4/3$ .

**CORRECT**



# Kahan on the computation of powers

“Nobody knows how much it would cost to compute  $y^w$  correctly rounded for every two floating-point arguments at which it does not over/underflow... No general way exists to predict how many extra digits will have to be carried to compute a transcendental expression and round it correctly to some preassigned number of digits. Even the fact (if true) that a finite number of extra digits will ultimately suffice may be a deep theorem.” —*William Kahan*

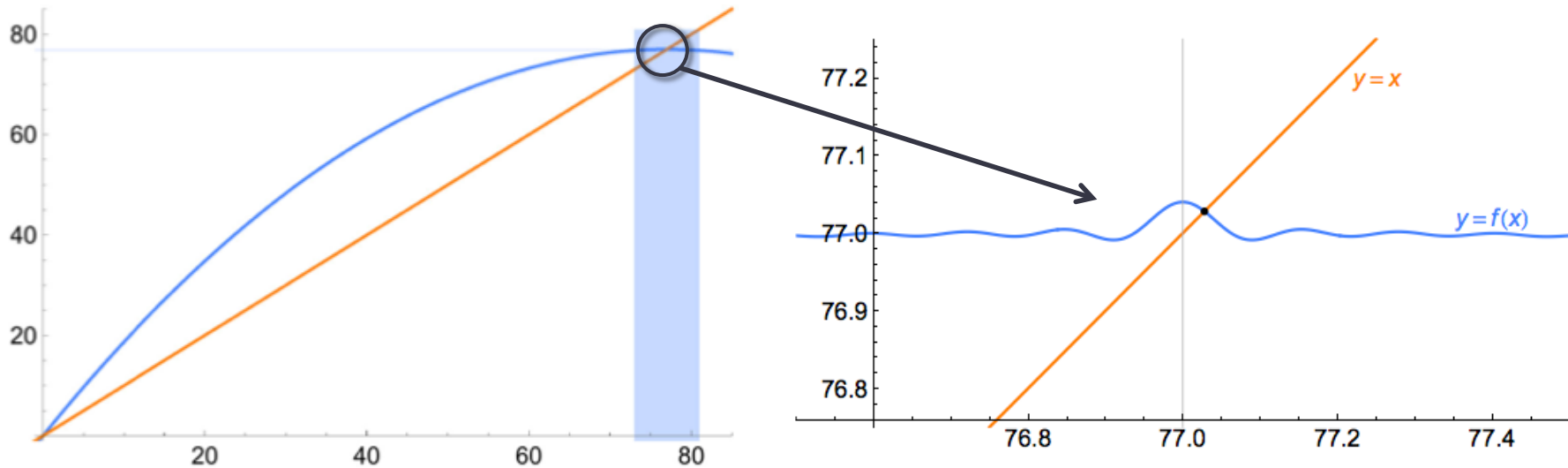
I sent Kahan my fixed-cost, fixed-storage method, and he said it looked “impractical.”

I asked if he had a method that shows the following computation is *exact*:

$$5.9604644775390625^{0.875} = 4.76837158203125$$

Have not heard from him since.

# Two can play this game, Professor K.



- Stable fixed point found by floats, not by traditional intervals
- Unums find both stable point, *and* unstable point at origin
- Finding exact stable point is mathematically incorrect!
- Adding a tiny wobble  $\sin(x)/x$  destroys floats, but not unums.

Unums can do anything floats can do, through explicit use of the **guess** function.

# Rump's Royal Pain

Compute  $333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$   
where  $x = 77617$ ,  $y = 33096$ .

- Using IBM (pre-IEEE Standard) floats, Rump got
  - 1.172603 in 32-bit precision
  - 1.1726039400531 in 64-bit precision
  - 1.172603940053178 in 128-bit precision
- Using IEEE double precision:  $1.18059 \times 10^{21}$
- **Correct answer:  $-0.82739605994682136\dots$ !**  
Didn't even get *sign* right

Unums: **Correct answer** to 23 decimals using an average of only **75** bits per number. Not even IEEE 128-bit precision can do that. Precision, range adjust *automatically*.

# Some fundamental principles

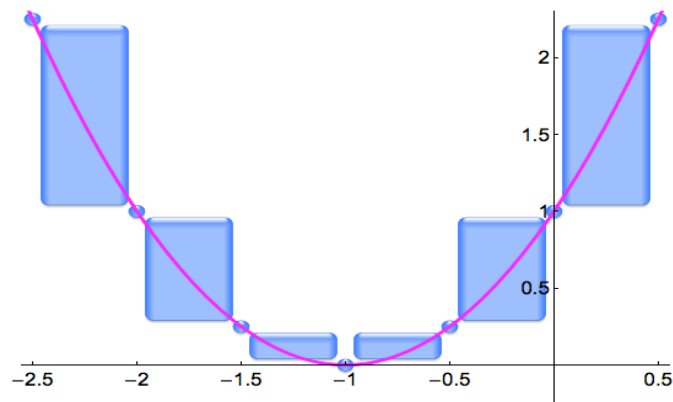
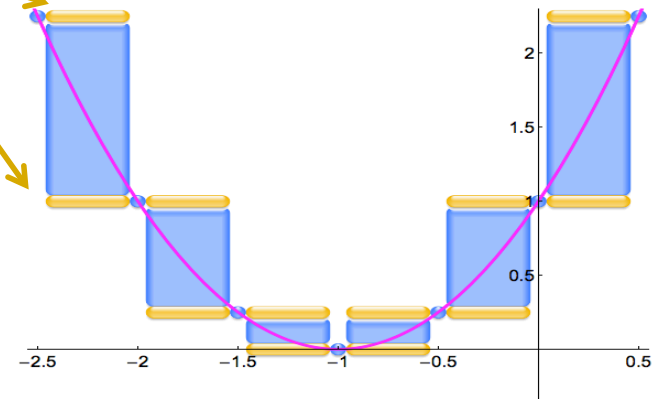
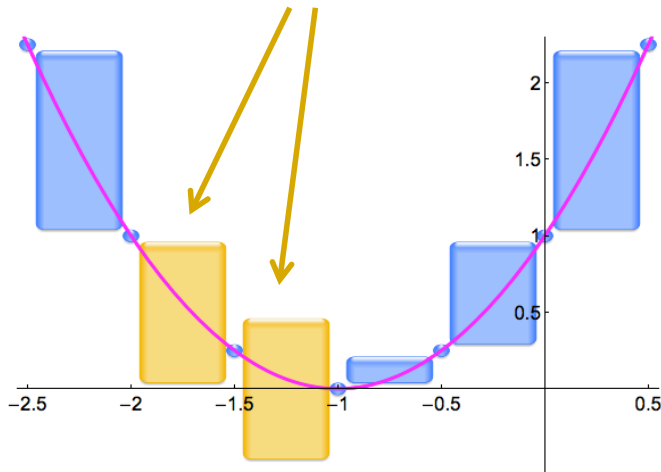
**Bound the answer as tightly as possible within the numerical environment, or admit defeat.**

- No more guessing
- No more “the error is  $O(h^n)$ ” type estimates
- The smaller the bound, the greater the information
- *Performance is information per second*
- Maximize information per bit
- Fused operations are always explicitly distinct from their non-fused versions and results are identical across platforms



# Polynomials: bane of classic intervals

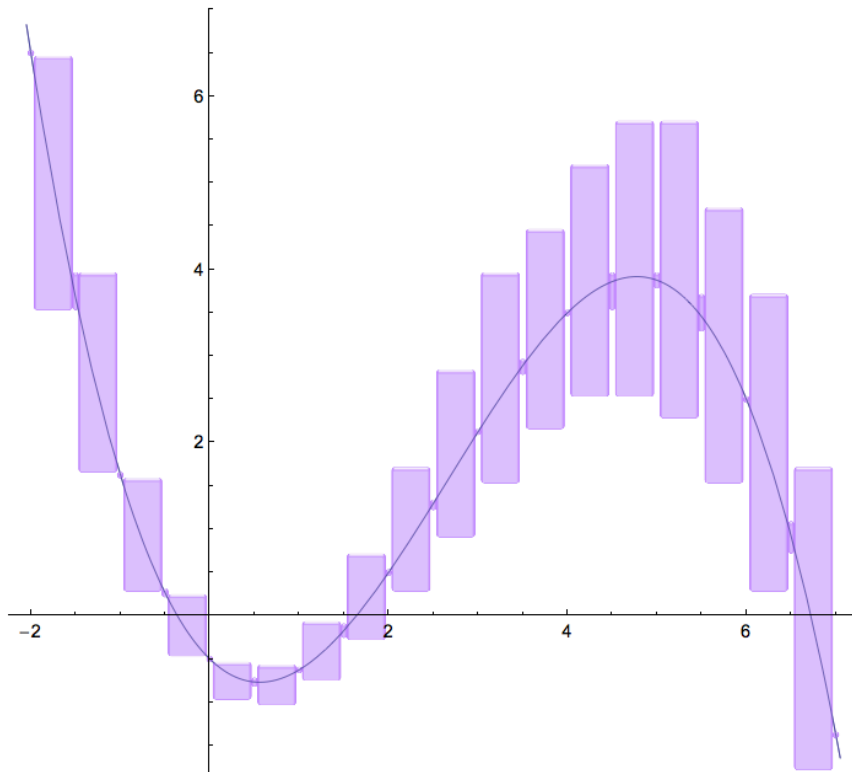
Dependency and closed endpoints lose information (amber)



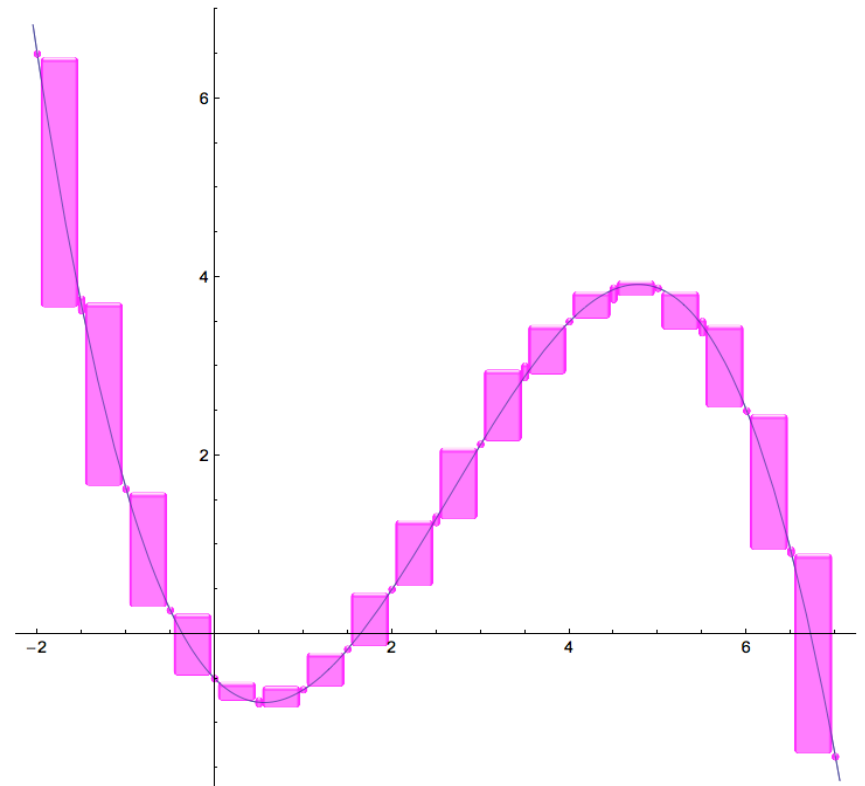
Unum polynomial evaluator  
loses *no* information.

# Polynomial evaluation solved at last

Mathematicians have sought this for at least 60 years.



“Dependency Problem” creates sloppy range when input is an interval



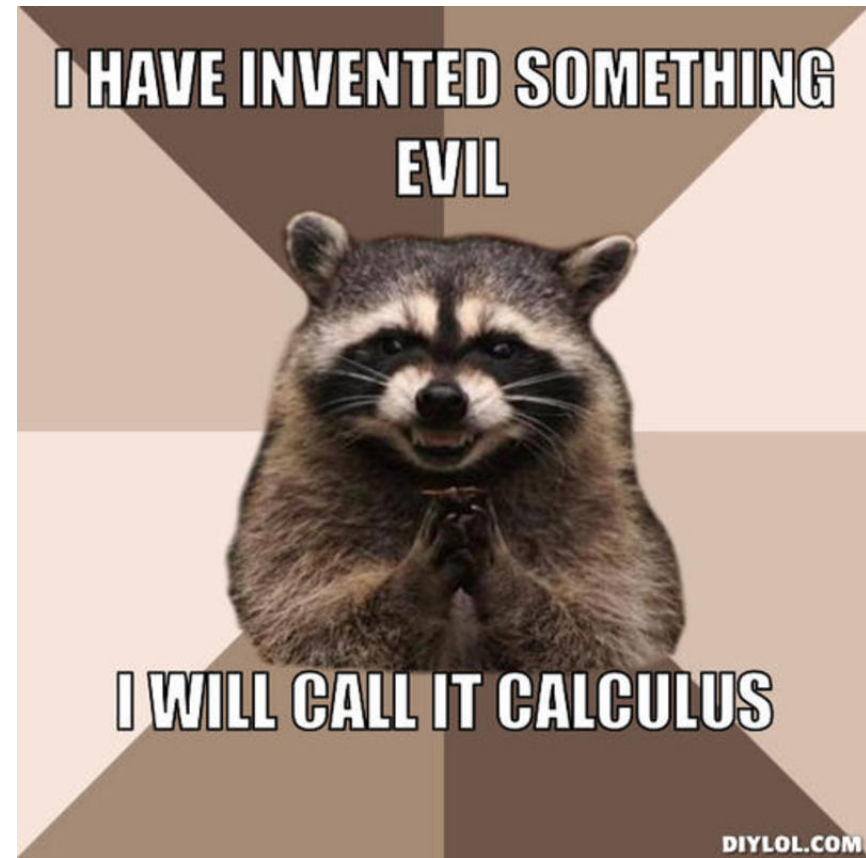
Unum evaluation refines answer to limits of the environment precision

# Uboxes and solution sets

- A *ubox* is a multidimensional unum
- Exact or ULP-wide in each dimension
- Sets of uboxes constitute a *solution set*
- One dimension per degree of freedom in solution
- Solves the main problem with interval arithmetic
- Super-economical for bit storage
- Data parallel in general

# Calculus considered harmful

- Computers are *discrete*
- Calculus is *continuous*
- Ensures sampling errors
- Changes problem to fit the tool

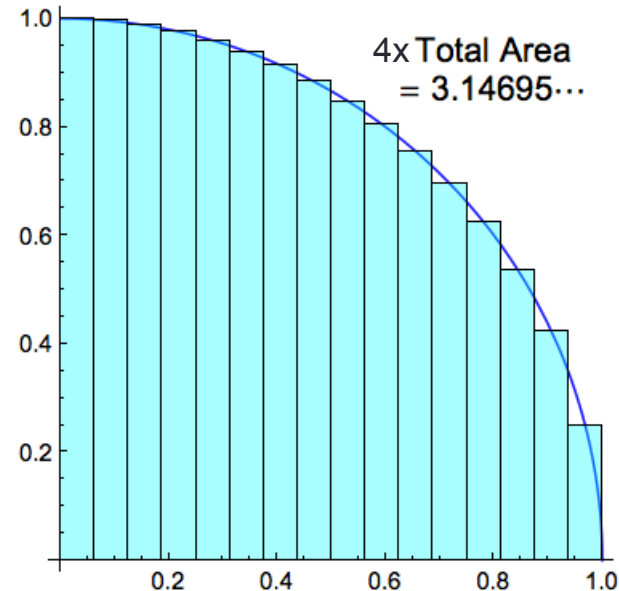


# Deeply Unsatisfying Error Bounds

- Classical numerical texts teach this “error bound”:

$$\text{Error} \leq (b - a) h^2 |f''(\xi)| / 24$$

- What is  $f''$ ? Where is  $\xi$ ?  
What is the bound??
- Bound is often *infinite*, which means no bound at all
- “Whatever it is, it’s four times better if we make  $h$  half as big” creates demand for supercomputing that *cannot be satisfied*.



# Two “ubox methods”, both mindless

- **Paint bucket:** find one solution point, test neighbors and classify as solution or fail until no more neighbors to test
  - Works if solution is known to be a connected set
  - Requires a starting point “seed”
  - Wave front of trial uboxes can be computed **in parallel**
- **Try the universe:** Use Warlpiri uboxes (4-bit precision) to tile all of  $n$ -space; increment exponent and fraction size automatically
  - $13^n$  things to do in parallel (!)
  - Finds *every* solution, no matter what, since all of  $n$ -space is represented
  - Detects ill-posed problems and solves them anyway
  - Parallelism adjusts from 3 to *trillions*



# Quarter-circle example

- Suppose all we know is  $x^2 + y^2 = 1$ , and  $x$  and  $y$  are  $\geq 0$
- Suppose we have at most 2 bits exponent, 4 bits fraction

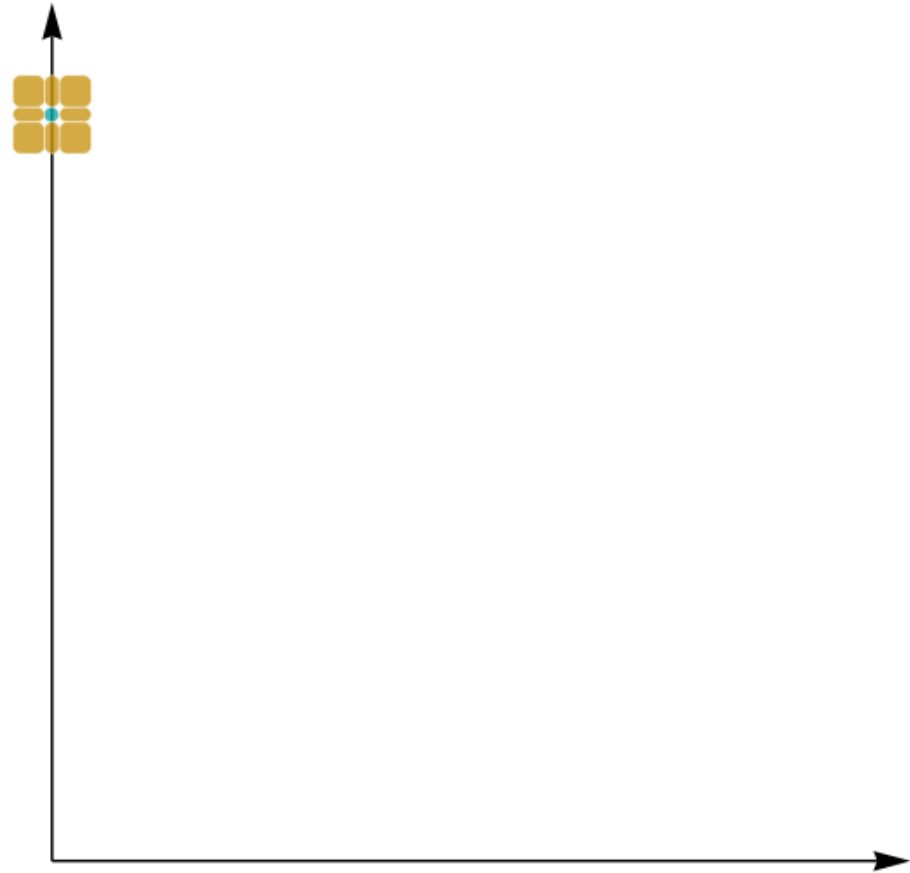
Task:

Bound the quarter circle area.

Bound the value of  $\pi$ .

# Set is connected; need a seed

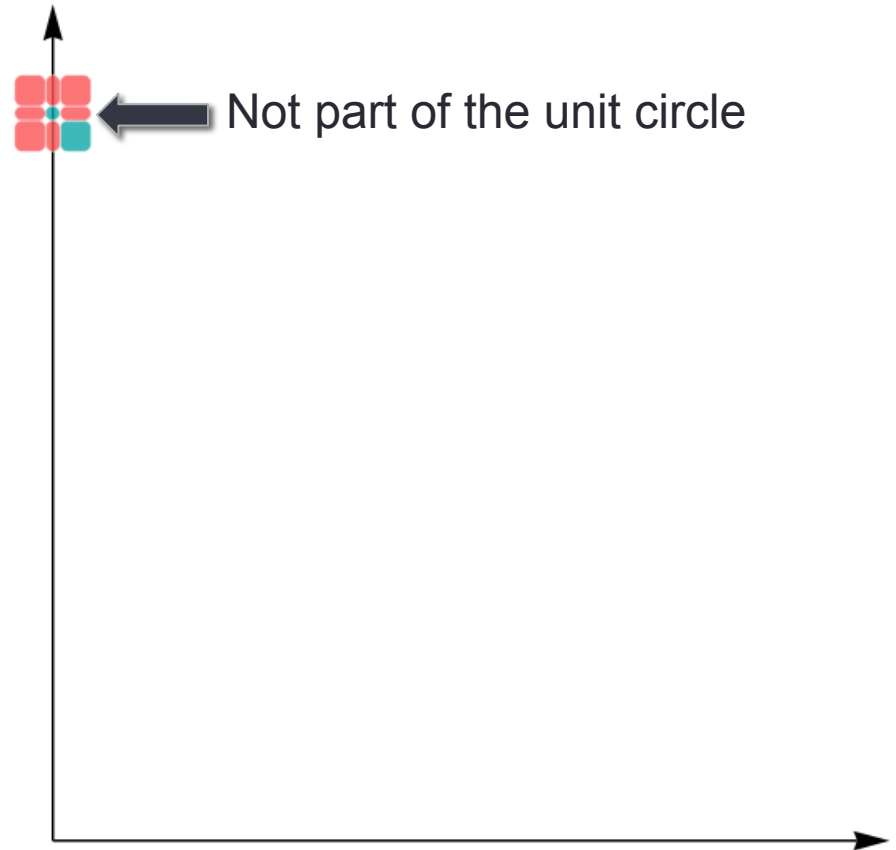
- We know  $x = 0, y = 1$  works
- Find its eight ubox neighbors in the plane
- Test  $x^2 + y^2 = 1, x \geq 0, y \geq 0$
- Solution set is green
- Trial set is amber
- Failure set is red
- Stop when no more trials



# Exactly one neighbor passes

- Unum math automatically excludes cases that floats would accept
- **Trials** are neighbors of new solutions that
  - Are not already **failures**
  - Are not already **solutions**
- Note: *no* calculation of

$$y = \sqrt{1 - x^2}$$



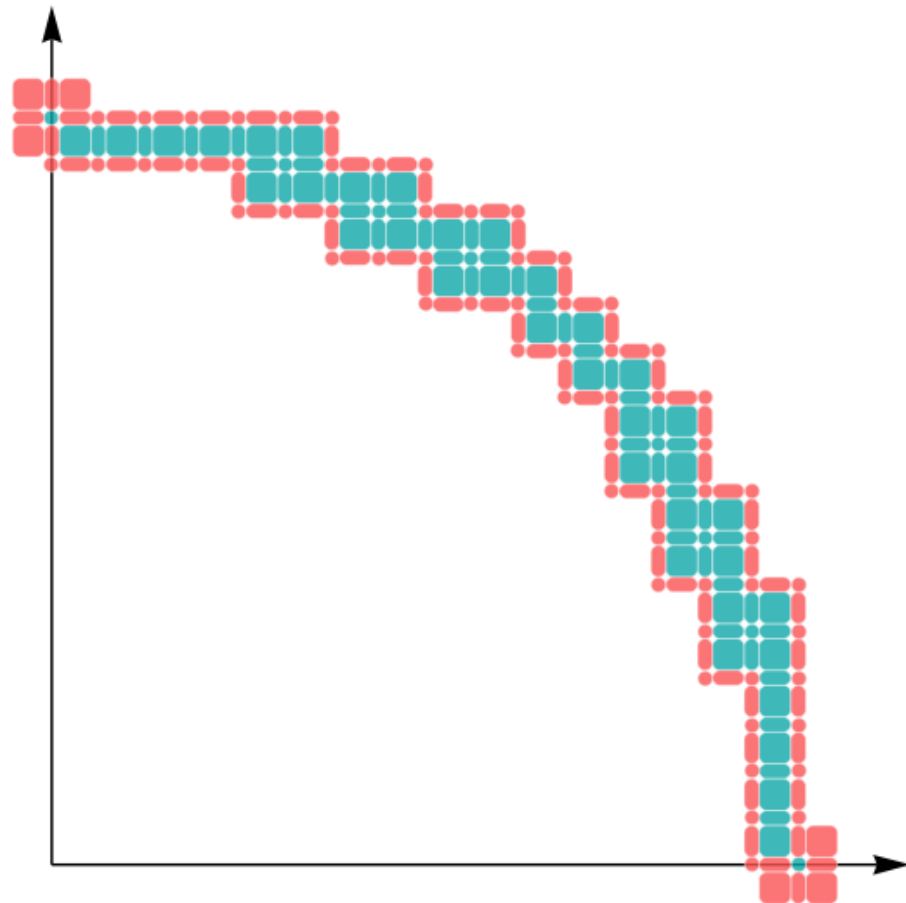
# The new trial set

- Five **trial** uboxes to test
- Perfect, easy parallelism for multicore
- Each ubox takes only 15 to 23 bits
- Ultra-fast operations
- Skip to the final result...



# The complete quarter circle

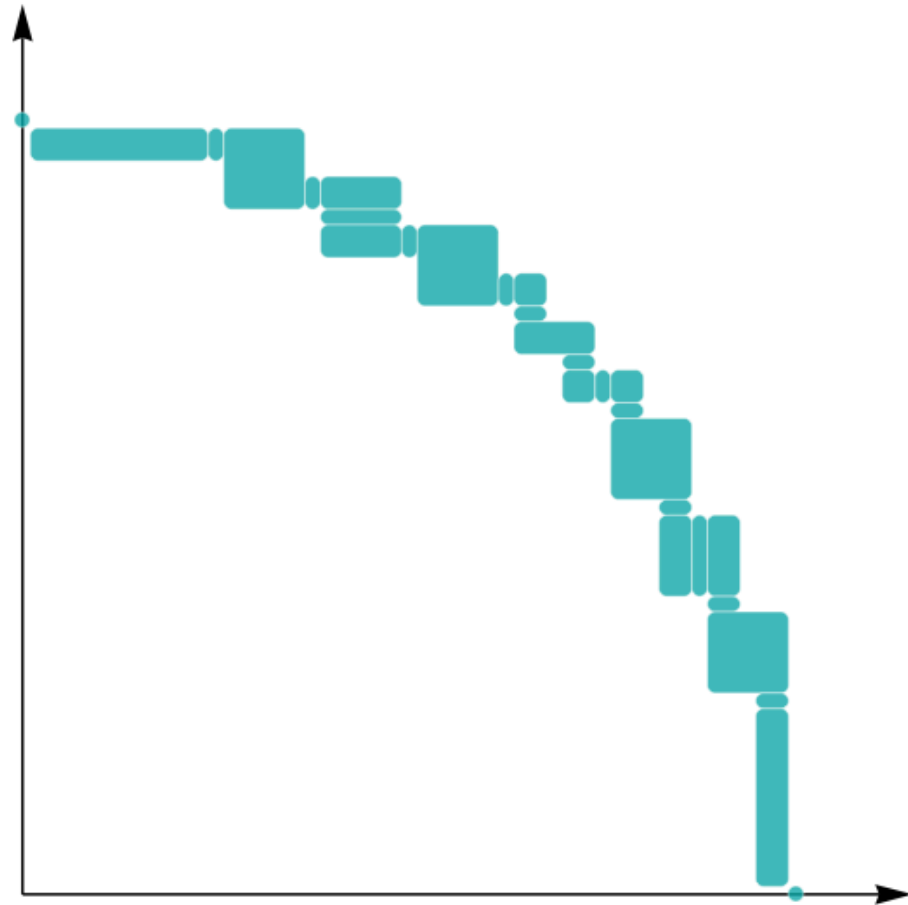
- The complete *solution*, to this finite precision level
- *Information* is reciprocal of green area
- Use to find area under arc, bounded above and below
- *Proves* value of  $\pi$  to an accuracy of 3%
- No calculus needed, or divides, or square roots



# Compressed Final Result

- Coalesce uboxes to largest possible ULP values
- *Loss/ess* compression
- Total data set: 603 bits!
- 6x faster graphics than current methods

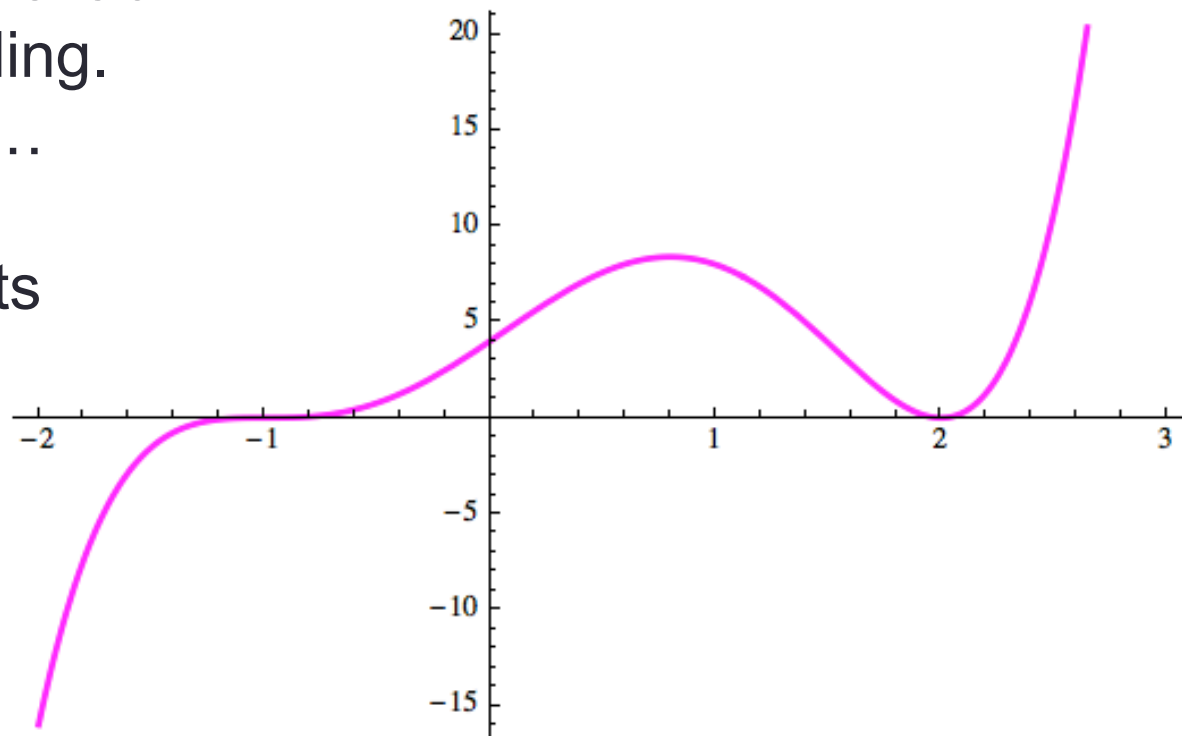
Instead of ULPs being the source of error, they are the *atomic units of computation*



# Fifth-degree polynomial roots

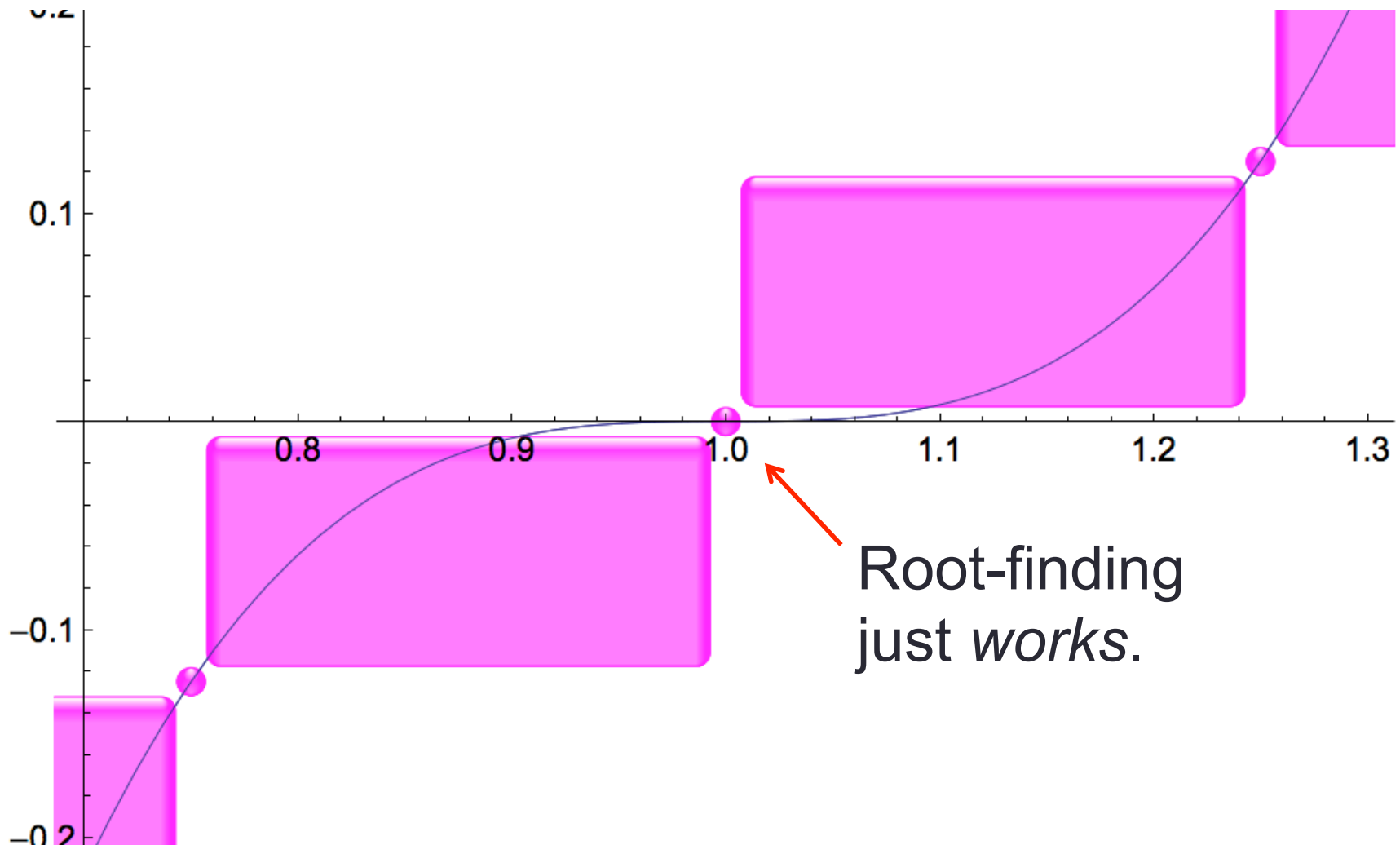
- Analytic solution: There isn't one.
- Numerical solution: Huge errors from rounding
- Unums: quickly return  $x = -1$ ,  $x = 2$  as the exact solutions. No rounding. No underflow. Just... the *correct answer*. With as few as 4 bits for the operands!

$$y = x^5 - x^4 - 5x^3 + x^2 + 8x + 4$$



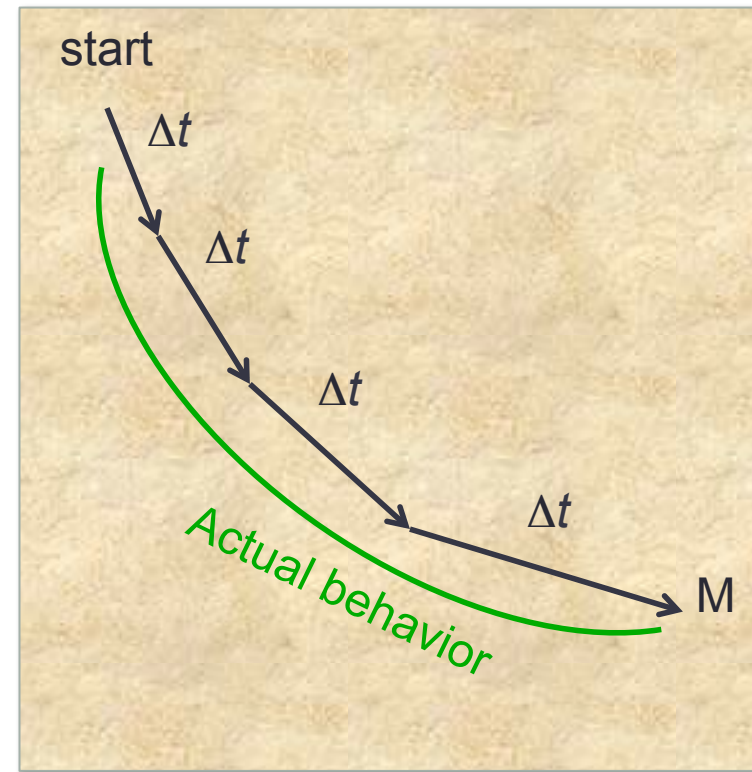


# The power of open-closed endpoints



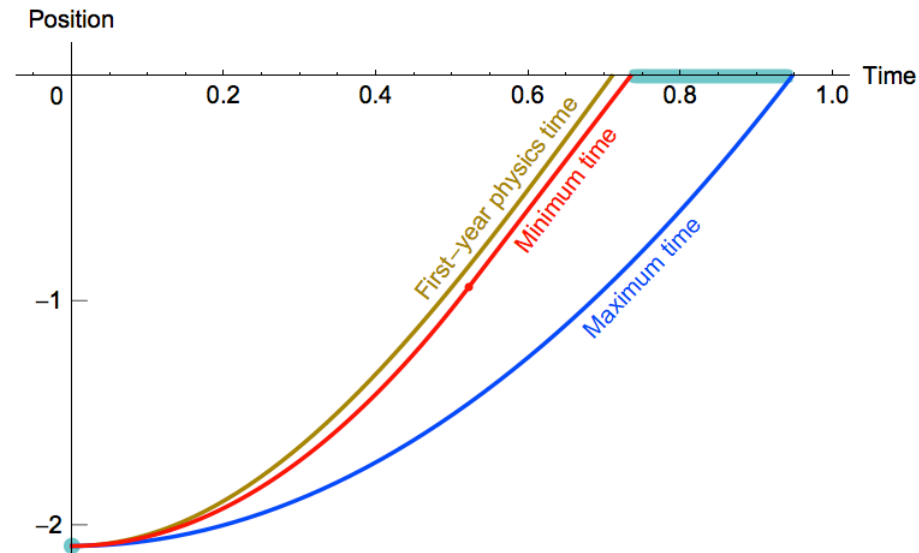
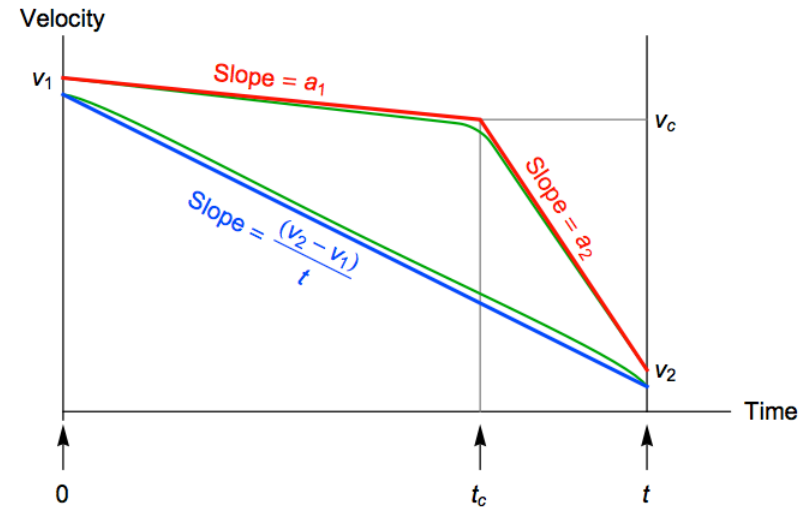
# Classical Numerical Analysis

- Time steps
  - Use position to estimate force
  - Use force to estimate acceleration
  - Update the velocity
  - Update the position
  - Lather, rinse, repeat
- Accumulates rounding and sampling error, both unknown
- ***Cannot be done in parallel***



# A New Type of Parallelism

- *Space* steps, not time steps
- Acceleration, velocity bounded in any given space interval
- Find traversal time as a function of space step (2D ubox)
- Massively parallel!
- *No rounding error*
- *No sampling error*
- Obsoletes existing ODE methods

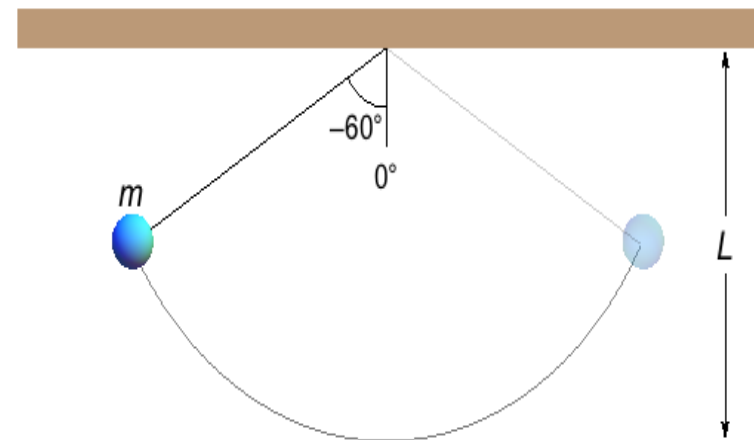


# Pendulums Done Right

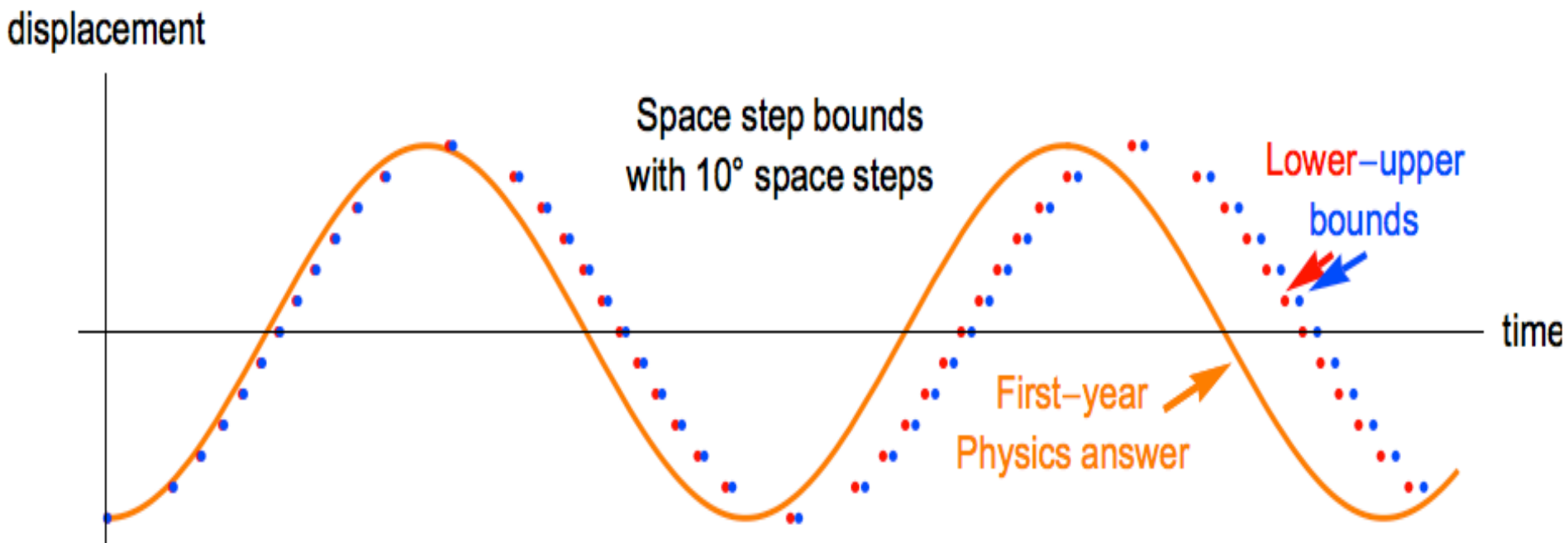
- Physics teaches us it's a harmonic oscillator with period

$$2\pi\sqrt{\frac{g}{L}}$$

- Force-fits nonlinear ODE into linear ODE for which calculus works.
- WRONG** answer



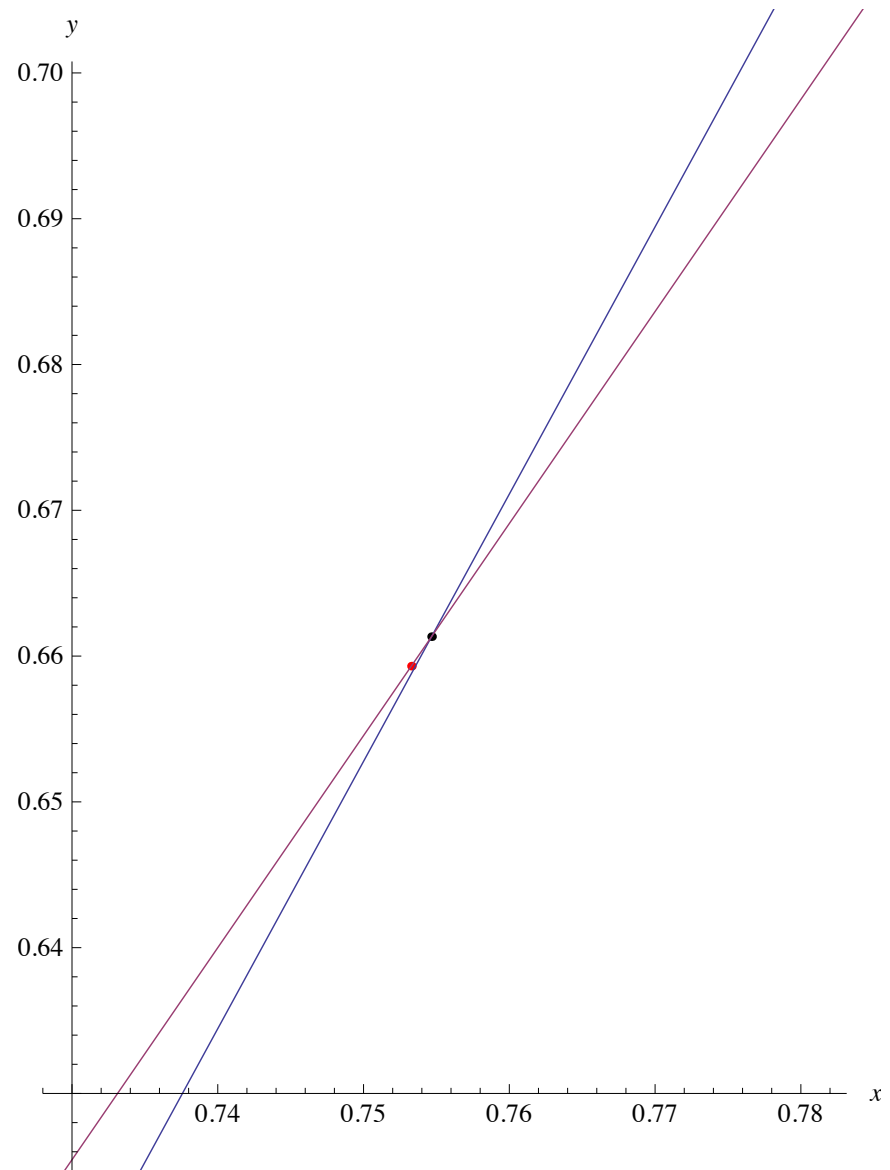
# Physical Truth vs. Force-Fit Solution



Bends the problem to fit solution methods

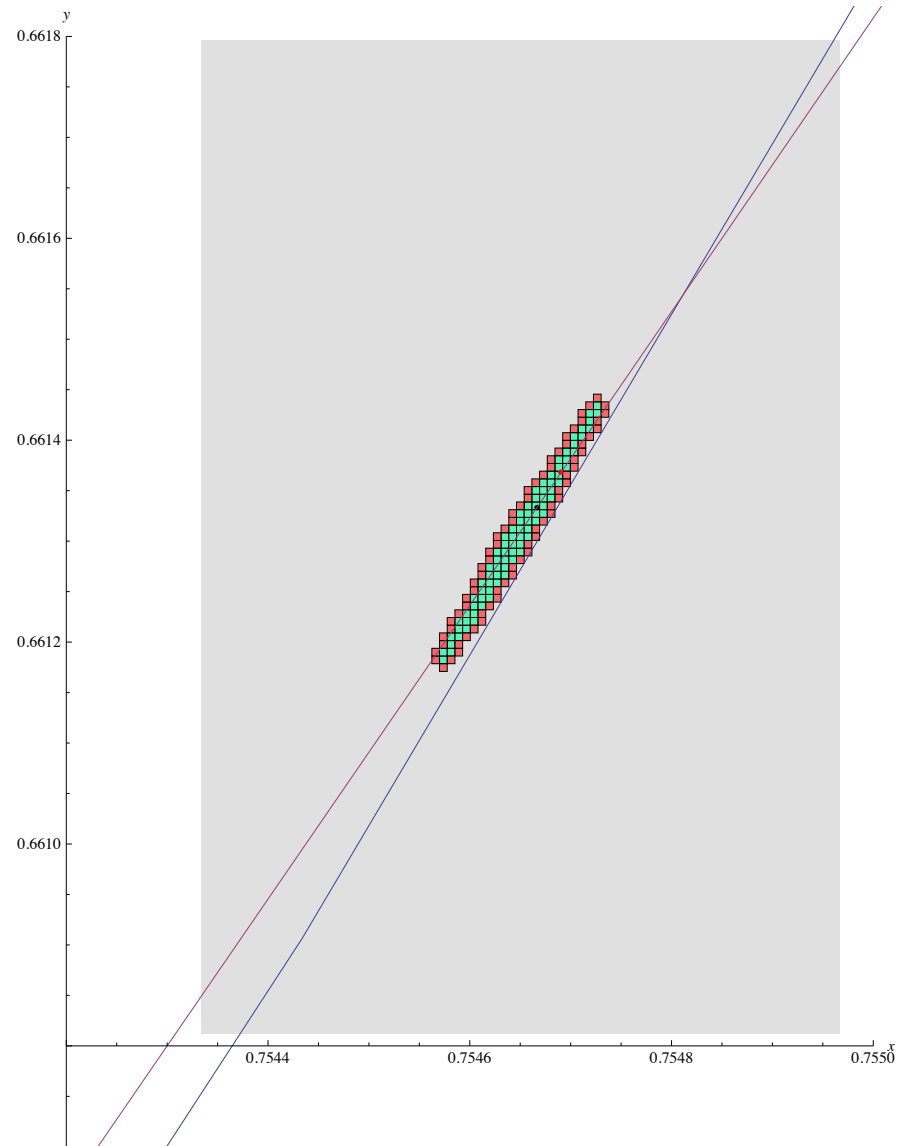
# Uboxes for linear solvers

- If the  $A$  and  $b$  values in  $Ax=b$  are rounded, the “lines” have width from uncertainty
- Apply a standard solver, and get the red dot as “the answer”,  $x$ . A pair of floating-point numbers.
- Check it by computing  $Ax$  and see if it rigorously contains  $b$ . Yes, it does.
- Hmm... are there any *other* points that also work?



# Float, Interval, and Ubox Solutions

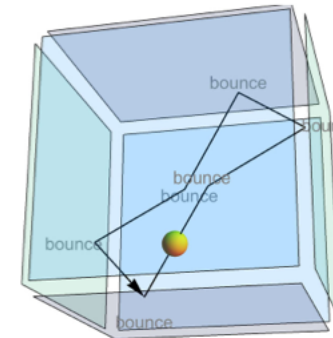
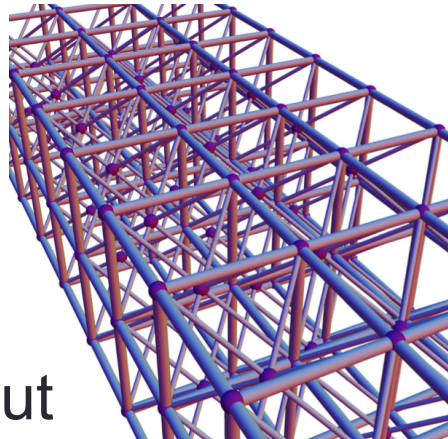
- Point solution (black dot) just gives *one of many* solutions; disguises answer instability
- Interval method (gray box) yields a bound too loose to be useful
- The ubox set (green) is the *best you can do for a given precision*
- Uboxes *reveal* ill-posed nature... yet provide solution anyway
- Works equally well on *nonlinear* problems!



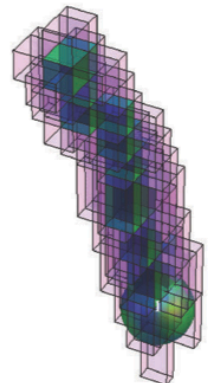
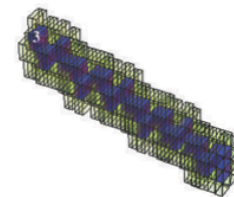
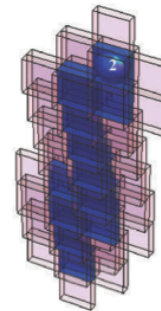


# Other Apps with Ubox Solutions

- Photorealistic computer graphics
- N-body problems
- Structural analysis
- Laplace's equation
- Perfect gas models without *statistical* mechanics



Imagine having **provable bounds** on answers for the first time, yet with easier programming, less storage, less bandwidth use, less energy/power demands, *and* abundant parallelism.



# Revisiting the Big Challenges-1

- *Too much energy and power needed per calculation*
  - Unums cut the main energy hog by about 50%
- *More hardware parallelism than we know how to use*
  - Uboxes reveal vast sources of *data* parallelism, the easiest kind
- *Not enough bandwidth (the “memory wall”)*
  - More use of CPU transistors, fewer bits moved to/from memory
- *Rounding errors more treacherous than people realize*
  - Unums eliminate rounding error, automate precision choice
- *Rounding errors prevent use of multicore methods*
  - Unums restore algebraic laws, eliminating the deterrent

# Revisiting the Big Challenges-2

- *Sampling errors turn physics simulations into guesswork*
  - Uboxes produce provable *bounds* on physical behavior
- *Numerical methods are hard to use, require expertise*
  - “Paint bucket” and “Try the universe” are brute force general methods that need no expertise... not even calculus



# The End of Error

- A complete text on unums and uboxes is available from CRC Press as of this month:  
<http://www.crcpress.com/product/isbn/9781482239867>
- Aimed at *general* reader; mathematicians will hate its casual style
- Complete prototype environment is available as *Mathematica* notebook through publisher

**Thank you!**

